

# **CN-Linux (Compute Node Linux)**

Alex Nagelberg

August 4, 2006

## **Contents**

<b>Abstract</b>	<b>2</b>
<b>Introduction</b>	<b>2</b>
<b>Overview of the BlueGene/L Architecture</b>	<b>3</b>
<b>ZOID (Zepto OS I/O Daemon)</b>	<b>4</b>
<b>CN-Linux (Compute Node Linux)</b>	<b>4</b>
<b>Performance</b>	<b>5</b>
<b>Future Work</b>	<b>5</b>
<b>Acknowledgements</b>	<b>5</b>
<b>Performance Figures</b>	<b>6</b>
<b>References</b>	<b>8</b>

## **Abstract**

BlueGene/L [1] (BG/L) is a supercomputer engineered by IBM researchers with space-saving and low power-consumption in mind. BG/L's architecture utilizes many nodes containing few components, which interact with each other to perform a given task. The nodes are divided in to 3 parts: the compute nodes, which handle computation; the I/O nodes, which route all input/output system calls appropriately to/from compute nodes; and the service nodes, which are regular computer systems that manage hardware resources and perform job control. There is currently no simple solution for compiling a large project on the BG/L. IBM has written its own proprietary operating system to run a process on a compute node (known as BLRTS). Simply compiling GCC (GNU Compiler Collection) for the BLRTS OS returned errors with memory allocation. Popular compilers also usually require multiple processes (to invoke compilation to assembly, compilation of assembly, and linking) as well as shared libraries in which BLRTS does not allow. These issues could be resolved using a Linux-based operating system, such as ZeptoOS, [2] under development at Argonne. Linux is now functional on compute nodes using a hack to execute the kernel under BLRTS. Libraries were written to initialize and perform communication of system calls to the I/O nodes by porting hacked GLIBC BLRTS code.

## **Introduction**

Modern mathematics and sciences have always had a great need for computational power. Currently, supercomputers and computer clusters are the most promising solutions for the need of heavy computational power. Because manufacturers

of supercomputers are in such a specialized field, the operating systems are sometimes limited to functionality that only the manufacturer can provide. IBM's BlueGene/L is one such system with part of its architecture using proprietary software. The ZeptoOS project [2] hopes to offer an alternative with useful tools to IBM's proprietary software.

### **Overview of the BlueGene/L Architecture**

The BlueGene/L [1] is divided up in to three parts: the compute nodes, the I/O nodes, and the service nodes. The compute nodes handle actual computation. Most system calls and I/O functions are routed through the I/O nodes. The pset of the BlueGene/L specifies how many compute nodes are assigned to a single I/O node. System calls get passed over the tree network to the I/O node assigned. The service nodes are regular computers. They simply drive the compute and I/O nodes, booting them up appropriately.

IBM's official packages provide kernels and ramdisk images to boot the compute and I/O nodes. The I/O nodes run a build of Linux with CIOD (Compute I/O Daemon) started up upon booting to listen and perform I/O operations for the compute nodes. The compute nodes run a small operating system, BLRTS (BlueGene/L Run-time Supervisor), written by IBM, which executes the job submitted on the compute nodes.

Because of BLRTS' slim design, it has several drawbacks. Firstly, BLRTS only allows one process to run. This can be problematic if one's job needs to launch other programs or if the program is multi-threaded. BLRTS also doesn't allow shared libraries, so one's program must be statically compiled. In instances where a program might need to use one of these major features, an alternative OS, such as Linux, would be necessary for the job.

### **ZOID (Zepto OS I/O Daemon)**

The standard way of handling I/O operations on BlueGene/L involves system calls invoked in BLRTS to be communicated over the tree network to the I/O node's daemon, CIOD. ZOID, part of the Zepto OS project [2], offers an open-source alternative for CIOD. ZOID currently works by using CIOD to initialize communication, and then suspends it to intercept messages from the compute nodes. It hopes to have multi-threading support finished, offering a performance advantage over CIOD.

### **CN-Linux (Compute Node Linux)**

The Linux kernel was successfully booted on the compute node through a hack provided by Kazutomo Yoshii. A program was written for BLRTS to load the Linux kernel and a ramdisk image in to memory, then execute the kernel leaving BLRTS in the dust acting as a boot-loader for the kernel under BLRTS. A tree device driver was also added in to the kernel allowing for communication over the tree network.

Support for system calls over the tree network to the I/O daemon is currently done through a separate library. Code was taken from the hacked GLIBC for BLRTS, and then RTS function calls removed and replaced as well as fixed memory addresses were replaced with dynamic ones. Problems were encountered trying to integrate this code with GLIBC (GNU C-library) due to the tree device existing on the compute node's local filesystem. Debugging was also an issue, since most error messages are displayed through use of `printf()` calls and `printf()` was only implemented to output to RAS; messages would arrive 10 minutes after the job ran. Initialization for GLIBC is problematic since, to enable functions such as `read()`, `write()`, etc, it must perform these functions that are being implemented to read/write to the tree device, which exists on the

local compute node filesystem. Once that is completed, those system calls will seamlessly be handled by the I/O daemon.

### **Performance**

Performance was measured using a simple program [3] to calculate read/write speed from the /dev/zero device (existing on the I/O node). The tests were performed using a single pset (only one I/O node) and were tested on 1-32 compute nodes, using buffer sizes ranging from 1 kilobyte to 8,192 kilobytes. The I/O node ran ZOID and Linux on the compute nodes' read/write speed was compared to that of IBM's BLRTS (see page 6). The throughput of CN-Linux was better than that of BLRTS, though it's not obvious why that is.

### **Future Work**

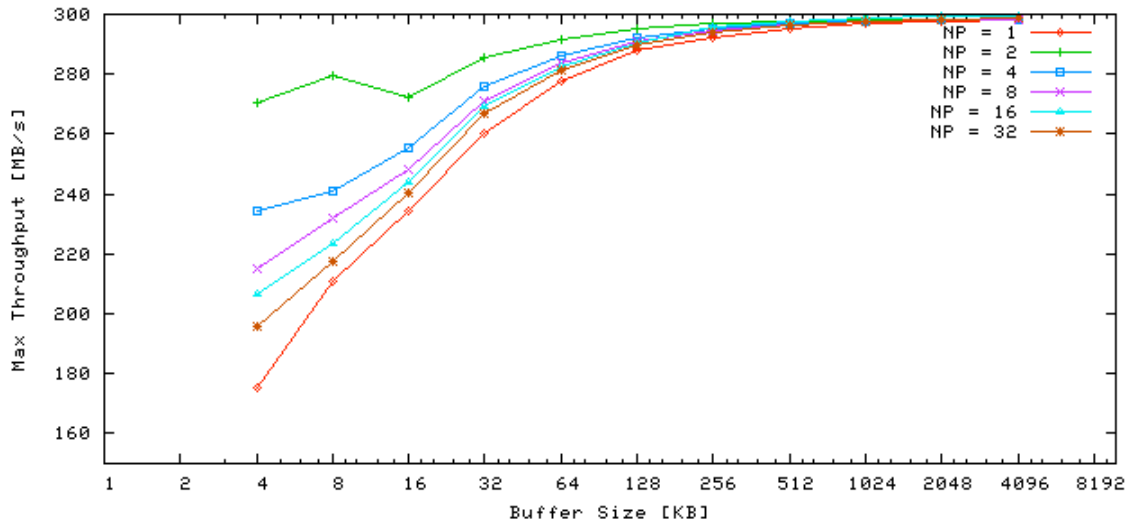
In the future, Zepto OS hopes to have full GLIBC support for the I/O nodes working on CN-Linux, as well as some method of MPI communication with the other compute nodes, and a full ramdisk image with tools for the compute nodes running Linux. ZOID will be multi-threaded in the near future, improving the I/O performance significantly.

### **Acknowledgements**

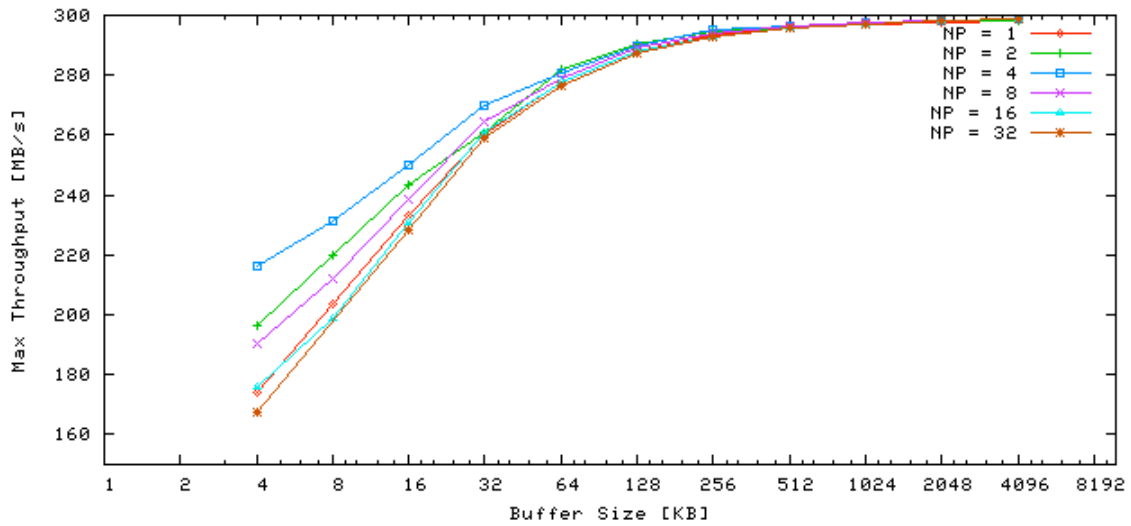
I'd like to thank the U.S. Department of Energy, the Office of Science, the SRP Program, and the Center of Computation and Technology at Louisiana State University for granting me this opportunity to research at Argonne National Laboratory. I'd like to extend especial thanks to Kamil Iskra and Kazutomo Yoshii for offering guidance and assistance in my research at Argonne. Finally, I'd like to thank Pete Beckman for having me on the Zepto OS team.

# Read Performance

CN-Linux

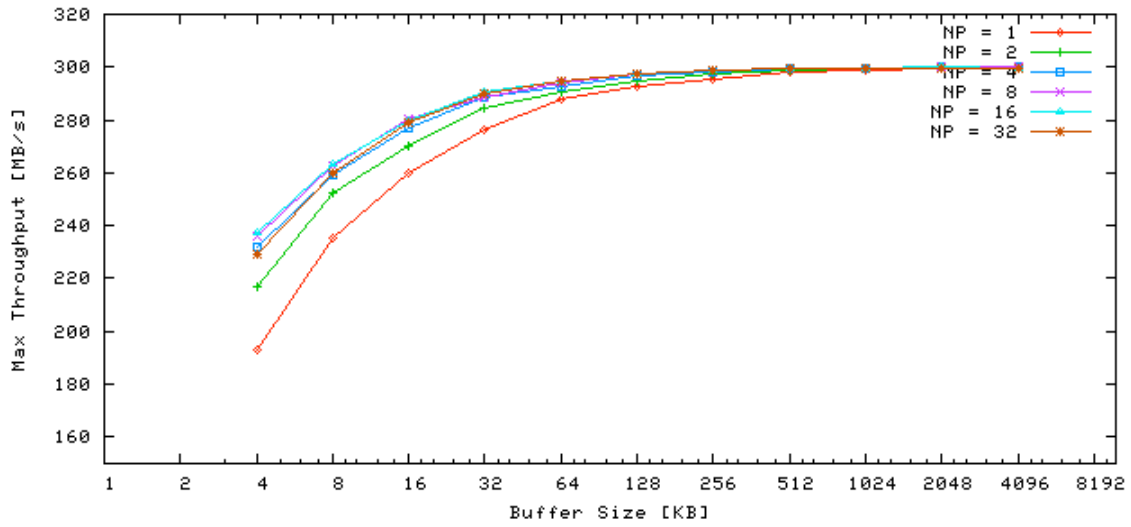


BLRTS

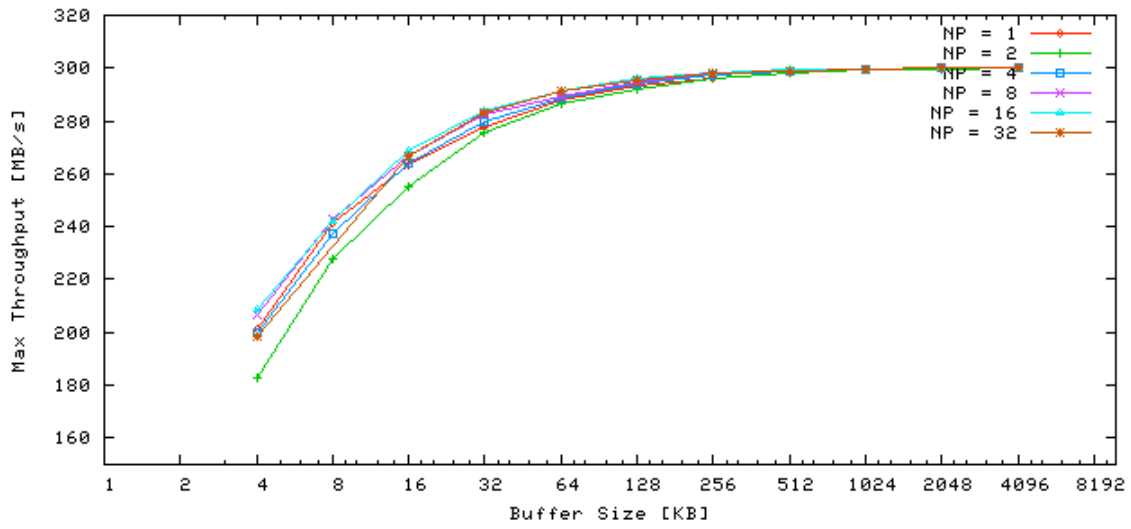


# Write Performance

CN-Linux



BLRTS



## References

- [1] A. Gara, et al. "Overview of the Blue Gene/L system architecture." In IBM Journal of Research and Development, Vol. 49, 2005, pp. 195-212
- [2] "ZeptoOS Project." <http://www.zeptoos.org>
- [3] "Unix I/O performance on Blue Gene/L." <http://www-unix.mcs.anl.gov/~iskra/bgl-unixio/>